**ABSTRACT**

We proposed a data mining approach for detecting malicious transactions in a Database System. The approach concentrates on mining data dependencies among data items in the database. A data dependency miner is designed for mining data correlations from the database log. The transactions not compliant to the data dependencies mined are identified as malicious transactions.

**INTRODUCTION**

Although many different approaches are employed to protect important data in today’s networked environment, these methods often fail. One way to make data less vulnerable is to deploy Intrusion Detection System (IDS) in critical computer systems. In case a computer system is compromised, an early detection is the key for recovering lost or damaged data without much complexity. In recent years, researchers have proposed a variety of approaches for increasing the intrusion detection efficiency and accuracy. But most of these efforts concentrated on detecting intrusions at the network or operating system level. They are not capable of detecting malicious data corruptions, i.e., what particular data in the database are manipulated by which specific malicious database transaction(s). Without this information, fast damage assessment and recovery cannot be achieved.

We propose a model for detecting malicious transactions that are targeted at corrupting data. When an attacker or a malicious user updates the database, the resulting damage can spread very quickly to other parts of the database through valid users. Quick and accurate detection of a cyber attack on a database system is the prerequisite for fast damage assessment and recovery. Malicious transactions identified in this work can be used later by database damage evaluation and recovery procedures. Our approach concentrates on mining data dependencies among data items in the database. By data dependency we refer to the data access correlations betweenThe techniques employed use data mining approach to generate data dependencies among data items. These dependencies generated are in the form of classification rules, i.e., before one data item is updated in the database what other data items probably need to be read and after this data item is updated what other data items are most likely to be updated by the same transaction. .

**THE DATA DEPENDENCY MINER**

The data dependency miner performs the analysis of data dependencies among data items in the database.

**Data Dependency Terminologies**

Because our overall goal is to discover data dependencies that are related to the sequence of operations performed by transactions, we first define the sequence in our context.

**Definition 1:** A sequence is an ordered list of read and/or write operations. We denote a sequence s by <o1(d1), o2(d2), …, on(dn)>, where oi ∈ {r, w} and dk is a data item, 1 ≤ k ≤ n. D(s) represents the set of data items contained in the sequence, i.e., D(s) = {d1, d2, …, dn}. The support for a sequence is defined as the fraction of total transactions that contains this sequence.

Read sequence and write sequence are employed to define read and write dependencies respectively.

**Definition 2:** The Read Sequence of data item x is the sequence with the format <r(d1), r(d2), …, r(dn), w(x)> which represents that the transaction may need to read all data items d1, d2, …, dn in this order before the transaction updates data item x. It must be noted that each data item may have several read sequences each having different length. All these sequences together are called the Read Sequence Set of this data item.

The notation rs(x) is used to denote the read sequence set of data item x. For example, consider the following update statement in a transaction.

Update Table1 set x = a + b + c where d = 90;

In this statement, before updating x, values of a, b, c and d must be read and then the new value of x is calculated. So <r(a), r(b), r(c), r(d), w(x)> ∈ rs(x).

It must be noted that the database log only contains before and after images of x instead of the mathematical operation used for calculating x, i.e., x = a + b + c. The above example is only for illustrating the concept of read sequence. The database log containing the above transaction may actually look like:

T1: r(m), r(n), w(y), r(u), r(v), r(a), r(b), r(c), r(d) w(x), r(a), w(c), commit.

Before the write operation w(x), 8 data items have been read. Some of them may not have data dependencies with x, e.g., they are read by another SQL statement in the same transaction. This means the new value of x is not directly dependent on the values of all these 8 data items. Our goal is to determine that in order to update x, data items a, b, c, and d are most likely need to be read and are relevant for calculating the new value of x. It must be noted that the mining result may only illustrate a and b have data dependencies with x. This may happen when some other transaction only read values of a and b before updating x.

**Definition 3:** The Write Sequence of data item x is the sequence with the format < w(x), w(d1), w(d2), …, w(dn) > which represents that the transaction may need to write all data items d1, d2, …, dn in this order after the transaction updates data item x. It must be noted that each data item may have several write sequences each having different length. All these sequences together are called Write Sequence Set of this data item.

For example, consider the following update statements in one transaction.

Update Table1 set x = a + b + c where …

Update Table1 set y = x + u where …

Update Table1 set z = x + w + v where …

Using the above example, it can be noted that <w(x), w(y),w(z)> is one write sequence of data item x, that is <w(x), w(y),w(z)> ∈ ws(x), where ws(x) denotes the write sequence set of x.

**Definition 4:** The Weight of Data Dependency indicates to what extent a data item x depends on other data items, i.e., D(s) – x, in its read or write sequence s. It’s defined by the possibilities of reading (writing) these data items before (after) updating x. The notions rweight(x, D(s) - x) and wweight(x, D(s) - x) denote the weight of read dependency and write dependency respectively. A pre-set threshold is used to identify whether a dependency is weak or strong.

For example, suppose the predefined threshold for weight of data dependency is 40%. For the sequence <r(a), r(b), r(c), r(d), w(x)>, if the probability of reading {a, b, c, d} before x is updated is 75%, then rweight(x , {a, b, c, d}) is equal to 75%. Since this is larger than the threshold, we say, the dependency between x and {a, b, c, d} is strong.

**Table 1. Example Transactions for Mining Sequential Patterns**

| Trans. ID | Transaction Operations |
| --- | --- |
| 1 | r(7),r(1),r(6),w(5),r(1),w(4); |
| 2 | r(1),r(5),w(1),r(4),r(5),w(4); |
| 3 | r(7),r(6),r(2),w(4),r(7),r(3),w(6),r(1),r(6),w(2), r(3),r(5),r(2),w(5); |
| 4 | r(2),w(2),r(4),r(7),w(3), r(6),w(5), r(1), w(4); |
| 5 | r(5),r(3),r(6),w(7); |
| 6 | r(6),r(1),w(3),r(1),w(6),r(2),r(7),r(4),w(2); |
| 7 | r(2),r(5),w(6); |
| 8 | r(4),w(6); |
| 9 | r(6),r(5),w(5),r(3),r(4),w(4),r(3),w(7); |
| 10 | r(5),r(6),w(5),r(5),w(4); |

**4.2 The Methodology**

Because our method uses the sequences of operations, i.e., what sequence of read operations must be performed before an update operation and what sequence of write operations must be done after the same update operation, it is intuitively similar to the problem of sequential pattern mining. But, by only employing sequential pattern mining algorithm on database log, we can only get some sequential patterns consisting of mixed read and write operations and these sequential patterns mined don’t necessary reflect the essential data correlations in a database system. In addition, it is hard to apply these sequences mined directly for detecting malicious transactions. By carefully analyzing the problem encountered, we found that by designing a rule generation algorithm, the sequential pattern discovering algorithm can be utilized to generate the desired classification rules for our purpose.

We split the problem with discovering data dependencies into three steps, namely, sequential pattern discovery phase, read and write sequence set generation phase, and data dependency rules generation phase.

**Table 2. Sequential Patterns Mined**

| Sequential Patterns with support > 25% | Support |
| --- | --- |
| r(3) | 30% |
| w(6) | 40% |
| r(1), w(4) | 30% |
| r(2), w(2) | 30% |
| r(2), r(7) | 30% |
| r(4), w(4) | 30% |
| r(5), w(4) | 30% |
| r(5), w(5) | 30% |
| r(6), r(5) | 30% |
| r(6), w(5), w(4) | 40% |
| r(7), r(6), r(1) | 30% |
| r(7), r(6), w(4) | 30% |
| r(7), r(6), w(5) | 30% |

4.2.1 Sequential Pattern Discovery Phase

Consider the 10 example transactions as shown in Table 1. The r(x) and w(x) represent read and write operations respectively and, without loss of generality, integers are used to represent each data item in the database. With minimum support set to 25%, i.e., a minimum support of 3 transactions, Table 2 illustrates 13 desired sequential patterns that satisfy the support constraint. For example, sequential pattern <r(6), w(5), w(4)> is supported by transactions 1, 4, 9, and 10. An example of a sequence that does not satisfy minimal support is the sequence <r(2), w(2), r(7)> that is only supported by transaction 4. Some sequences, e.g., <r(1)> and <r(7), r(6)> are not in the answer set because they are not maximal although they have minimum support.

4.2.2 Read and Write Sequence Set Generation Phase

By observing the sequential patterns mined in Table 2, it’s clear that some patterns can be used for discovering data dependencies while others should not be considered. First, some sequential patterns mined only contain one operation. For example, the sequential pattern <r(3)> or <w(6)> only contain the operation on one data, so that no data dependency can be generated and they should not be considered. Second, some sequential patterns mined only contain read operations. Since we are mostly concerned about malicious modifications to data items by user transactions, we will only pay attention to the transactions containing write operation(s). So patterns containing only read operations should also be neglected.

For all other sequential patterns, the following procedures are employed for generating the read and write sequence sets. For each write operation w(di) in sequential patterns, add <r(di1),  r(di2), r(di3),…, r(din), w(di)> to the read sequence set of data item di where {r(di1), r(di2), r(di3),…, r(din)} is the set of all read operations before w(di). Similarly, add <w(di), w(dj1), w(dj2), w(dj3),…, w(djk)> to write sequence set of data item di where {w(dj1), w(dj2), w(dj3),…, w(djk)} is the set of all write operations after w(di).

Table 3 illustrates the read and write sequence sets generated by using the above method from the sequential patterns mined in Table 2. For example, the sequence <r(6), w(4)> denotes that before data item 4 is updated, data item 6 should be read. While the sequence <r(7), r(6), w(4)> represents that before data item 4 is updated, data item 7 and 6 should be read in sequence. Of these two sequences, the one that represents more accurate dependency can be determined by analyzing rweight(4, {6}) and rweight(4, {7, 6}) and this will be illustrated in the next sub-section. In the write sequence set, there is only one item <w(5), w(4)> that denotes that after data item 5 is updated, data item 4 should be updated.

**Table 3. Read and Write Sequence Set**

| Read Sequence Set | Write Sequence Set |
| --- | --- |
| r(1), w(4) | w(5), w(4) |
| r(2), w(2) |  |
| r(4), w(4) |  |
| r(5), w(4) |  |
| r(5), w(5) |  |
| r(6), w(5) |  |
| r(6), w(4) |  |
| r(7), r(6), w(4) |  |
| r(7), r(6), w(5) |  |

4.2.3 Data Dependency Rules Generation Phase

Data dependency rules are categorized as read rules and write rules. The following procedure is utilized to generate data dependency rules. For all sequential patterns <r(di1), r(di2), …, r(din), w(di) > in read sequence set, generate the read rules with the format w(di)→r(di1), r(di2), …, r(din). If the confidence of the rule is larger than the minimum confidence, then it’s added to the answer set of read rules which depicts that before updating di, data items di1, di2, …, din must be read by the same transaction. Similarly for all sequential patterns w(di), w(dj1), w(dj2), …, w(djk) in the write sequence set, generate the write rules with the format w(di)→w(dj1), w(dj2), …, w(djk). If the confidence of the rule is larger than the minimum confidence, then it’s added in the answer set of write rules which depicts after updating di, data items dj1, dj2, …, djk must be updated by the same transaction.

For example, both <r(6), w(4)> and <r(7), r(6), w(4)> belong to the read sequence set of data item 4. So two read rules A: w(4) → r(6) and B: w(4) → r(7), r(6) can be generated. But, it doesn’t reflect how strong the data correlation between data item 4 and {6} or between data item 4 and {7, 6} is, even though both of them satisfy the minimum support specified. Suppose the minimum confidence is set to 70%. Because of the confidence of rule B is 50%, i.e., rweight(4, {7, 6}) = 50%, it’s not in the answer rule set. Whereas confidence of rule A is 83%, rweight(4,{6}) = 83%, it’s selected. The read and write rules generated from the read and write sequence set in Table 3 are illustrated in Table 4. These rules work as classification rules for identifying malicious transactions in the database system.

**Table 4. Data Dependency Rules Generated**

| Data Dependency Rules | Confidence |
| --- | --- |
| w(2)→r(2) | 100% |
| w(5)→r(6) | 100% |
| w(4)→r(6) | 83% |
| w(5)→w(4) | 80% |

After the read and write rules generated, they can be used to detect malicious transactions by checking the database log. The transactions that made modifications to the database without following data dependency rules are flagged as malicious transactions. The procedure is as follows. For all database transactions that have write operation(s) in it, verify whether each write operation follows data dependency rules, i.e., whether this transaction read appropriate data items before this update operation and whether it wrote appropriate data items after this update operation.

Let’s walk through an example to illustrate the detection procedure. Suppose we have transaction T1: r(2), r(5), w(2), r(6), r(1), w(7), r(4), r(3), r(5), w(5). In T1, data items 2, 7, and 5 are updated. For data item 2, the rule w(2)→r(2) is satisfied because before data item 2 is updated, data item 2 itself is read by the transaction. For data item 7, because there’s no data dependency mined for it, it doesn’t need to be checked. For data item 5, the rule w(5)→r(6) is satisfied but the rule w(5)→w(4) is not because after data item 5 is updated, no other data items are updated by the same transaction. So transaction T1 is identified as a malicious transaction.

**THE ALGORITHM**

The formal algorithm for mining data dependency rules is presented next.

Algorithm:

1. Initialize the read sequence set RS ={} and the write sequence set WS= {}

2. Initialize the read rule set RR={} and the write rule set WR={}

3. Generate the sequential patterns X = {xi | support(xi) > minimum support} by using existing sequential pattern mining algorithm

4. **for** each sequential pattern xi where | xi | > 1

**if** there’s a write operation in it

**for** each write operation wi ∈ xi

**if** <r(di1), r(di2), r(di3),…, r(din), w(di)>∉ RS and < r(di1), r(di2), r(di3),…, r(din)> ≠ <∅>

add <r(di1), r(di2), r(di3),…, r(din), w(di)> to RS where r(di1), r(di2), r(di3),…, r(din)

are all read operations before w(di)

**if** <w(di), w(dj1), w(dj2), w(dj3),…, w(djk)> ∉ WS and <wj1, wj2, wj3,…, wjk> ≠ <∅>

add <w(di), w(dj1), w(dj2), w(dj3),…, w(djk)> to WS where w(dj1), w(dj2), w(dj3),…, w(djk) are all write operations after wi

5. **for** each sequence in RS

**if** support(<r(di1), r(di2), r(di3),…, r(din), w(di)>) /support(<wi(di)>) > minimum confidence add w(di)→r(di1), r(di2), …, r(din) to RR

**for** each sequence in WS

**if** support(<w(di), w(dj1), w(dj2), …, w(djk)>) /support(<wi(di)>) > minimum confidence

add w(di)→w(dj1), w(dj2), …, w(djk) to WR